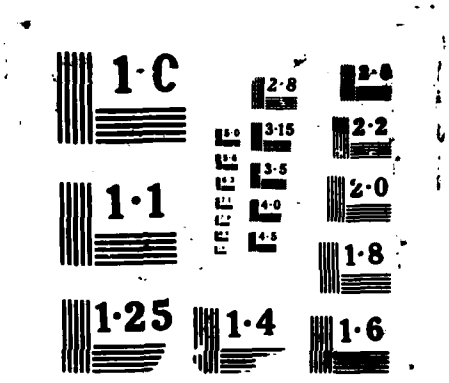
AD-A188 849 A SCHUR-COMPLEMENT METHOD FOR SPARSE QUADRATIC PROGRAMMING(U) STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB P E GILL ET AL OCT 87 SOL-87-12 N00014-87-K-0142 1/1

UNCLASSIFIED F/G 12/4 NL


1·C
1·1
1·25
1·4
1·6
2·8
3·15
3·5
2·2
2·0
1·8

Systems
Optimization
Laboratory

A Schur-Complement Method for Sparse Quadratic Programming

by
Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 87-12

October 1987

Department of Operations Research
Stanford University
Stanford, CA 94305

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

A Schur-Complement Method for Sparse Quadratic Programming

by
Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 87-12

October 1987

Research and reproduction of this report were partially supported by the National Science Foundation Grants CCR-8413211; U.S. Department of Energy Contract DE-FG03-87ER25030; Office of Naval Research Contract N00014-87-K-0142; Air Force Office of Scientific Research Grant AFOSR 87-01962.

A Schur-Complement Method for Sparse Quadratic Programming†

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright
Department of Operations Research
Stanford University
Stanford, California 94305-4022

Technical Report SOL 87-12
October 1987

Abstract

In applying active-set methods to sparse quadratic programs, it is desirable to utilize existing sparse-matrix techniques. We describe a quadratic programming method based on the classical Schur complement. Its key feature is that much of the linear algebraic work associated with an entire sequence of iterations involves a *fixed* sparse factorization. Updates are performed at every iteration to the factorization of a smaller matrix, which may be treated as dense or sparse.

The use of a fixed sparse factorization allows an off-the shelf sparse equation solver to be used repeatedly. This feature is ideally suited to problems with structure that can be exploited by a specialized factorization. Moreover, improvements in efficiency derived from exploiting new parallel and vector computer architectures are immediately applicable.

An obvious application of the method is in sequential quadratic programming methods for nonlinearly constrained optimization, which require solution of a sequence of closely related quadratic programming subproblems. We discuss some ways in which the known relationship between successive problems can be exploited,

† This paper is dedicated to the memory of James H. Wilkinson, whose unfailing insight and clarity of exposition remain an inspiration to us.

The material contained in this report is based upon research supported by the Air Force Office of Scientific Research Grant 87-01962; the U.S. Department of Energy Grant DE-FG03-87ER25030; National Science Foundation Grant CCR-8413211; and the Office of Naval Research Contract N00014-87-K-0142.

1. Background on Quadratic Programming

1.1. Statement of the problem. The topic of concern is the quadratic programming (QP) problem of minimizing a quadratic objective function subject to linear constraints on the variables. Quadratic programs may be stated in several (equivalent) forms. We shall consider primarily quadratic programs in the following *standard form*:

$$\begin{array}{ll} \underset{x \in \Re^n}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax = b, \qquad \ell \leq x \leq u, \end{array} \tag{1.1}$$

where the *Hessian matrix* H is symmetric and A is $m \times n$. Components of ℓ and u may be taken as $-\infty$ and $+\infty$ if no bound is present. We assume throughout that A has full row rank. The constraints $Ax = b$ are called the *general* constraints of (1.1). We define the (linear) function $g(x)$ as $c + Hx$, the gradient of the quadratic objective function.

The term "standard form" refers to the constraints in (1.1), and means that the only *inequality* constraints are *simple bounds on the variables.* (Section 6 treats some of the issues that arise with alternative formulations.) In much of our discussion, we shall treat all the variables of (1.1) uniformly. On some occasions, however, the "original" variables of a quadratic program will be distinguished from its "slack" variables. A quadratic program will contain slack variables if its "natural" formulation includes general inequality constraints. For example, a general inequality constraint $a_i^T x \geq \beta_i$ is replaced by the equality constraint $a_i^T x + s_i = \beta_i$, and the standard-form version of the problem includes an additional *slack variable* subject to the bound $s_i \leq 0$. Slack variables have many special features; one of particular importance is that they do not appear in the objective function. This paper is concerned only with problems in which the Hessian matrix *H in the original variables* is positive definite. The indefinite case will be treated in a forthcoming paper.

Our interest in sparse quadratic programs arises in large part from the desire to apply sequential quadratic programming (SQP) methods to large nonlinearly constrained problems. In an SQP method, each iteration involves solution of a quadratic programming subproblem, which itself must be solved by an iterative procedure. An important feature of these subproblems is that information from each can be exploited to solve the next more quickly, to the extent that later subproblems usually require only a single iteration (see Gill, Murray, Saunders and Wright, 1985). Thus, the *first* QP iteration comprises a substantial proportion of the total effort, which implies that *initialization* of the QP algorithm is just as critical as subsequent iterations.

Sections 2–5 describe a new method (the *Schur-complement* or SC method) for quadratic programming. Before giving details of the SC method, Section 1.2 introduces some notation, and Section 1.3 gives a condensed overview of active-set quadratic programming methods.

1.2. Notation. The proposed method is iterative, and we usually consider a single (typical) iteration. Unbarred and barred symbols will be used to denote quantities associated with iterations k and $k+1$. The only exception is the use of the suffix "0" to denote quantities associated with the first iteration.

We shall make extensive use of properties of the *inertia* of a matrix K, denoted by $\text{In}(K)$, which is an integer triple (α, β, γ), where α, β and γ are the numbers of positive, negative and zero eigenvalues of K.

Given a symmetric matrix

$$K = \begin{pmatrix} M & N^T \\ N & G \end{pmatrix},$$

with M nonsingular, the *Schur complement* of M in K will be denoted by K/M, and is defined as

$$K/M \equiv G - NM^{-1}N^T.$$

We sometimes refer simply to "the" Schur complement when the relevant matrices are clear. (For further discussion of the Schur complement, see Cottle, 1974.)

1.3. Background on active-set methods. The Schur-complement method is a *primal-feasible active-set method*. For an overview, see, e.g., Fletcher (1981). Each iteration has the following general structure: given the current iterate x, the next iterate is defined by

$$\bar{x} = x + \alpha p, \tag{1.2}$$

where the vector p is the *search direction*, and the nonnegative scalar α is the *steplength*. An initial *feasibility phase* is performed to find a point that satisfies the constraints of (1.1) (see Section 4), and all iterates are thereafter constructed to retain feasibility.

A major question in solving (1.1) is the identification of the *active set* of constraints, namely the constraints that hold with equality at the solution. Because (1.1) is in standard form, the active set must contain the general constraints, plus the set of variables that lie on one of their bounds at the solution. An active-set method maintains an estimate of the active set (called the *working set*), which is a linearly independent set of constraints that are satisfied exactly at the beginning of each iteration. The matrix of coefficients of constraints in the working set will be denoted by A_W, and always includes the equality constraints. Thus, a typical working set has the form

$$A_W = \begin{pmatrix} A \\ I_x \end{pmatrix},$$

where I_x contains rows of the identity corresponding to variables currently on their bounds. The constraints in the working set are (temporarily) treated as *equalities* during the current iteration.

The search direction p is defined as the solution of the following *equality-constrained* QP:

$$\begin{aligned} \underset{p \in \Re^n}{\text{minimize}} \quad & g^T p + \tfrac{1}{2} p^T H p \\ \text{subject to} \quad & A_W p = 0, \end{aligned} \tag{1.3}$$

where g denotes $g(x)$. The constraints $A_W p = 0$ ensure that constraints in the working set remain unaltered by any move along p. In particular, the components of p corresponding to bounds in the working set ("active" bounds) must be zero. The solution of (1.3) is the step from x to the minimizer of the quadratic objective function of (1.1), subject to treating the working set as equalities. The optimality and feasibility conditions for (1.3) are expressed by the linear system

$$\begin{pmatrix} H & A_W^T \\ A_W & \end{pmatrix} \begin{pmatrix} -p \\ \mu \end{pmatrix} = \begin{pmatrix} g \\ \end{pmatrix}, \tag{1.4}$$

where μ is the Lagrange multiplier vector for the constraints of (1.3).

Almost all active-set feasible-point methods for convex quadratic programming are mathematically identical in the sense that, under certain conditions, the same sequence of iterates is generated (see Djang, 1980, and Best, 1984). Differences in efficiency and numerical stability arise from the techniques chosen for solving (1.4). *Null-space methods* are based on computing either implicitly or explicitly a (nonunique) matrix Z whose columns span the null space of A_W. The solution of (1.4) can then be computed as $p = Zp_z$, where p_z satisfies

$$Z^T H Z p_z = -Z^T g. \tag{1.5}$$

The matrix $Z^T H Z$ is called the *projected Hessian*. We stress that the projected Hessian depends on the choice of A_W as well as on the representation of Z.

If p is nonzero, two situations are possible. The point $x+p$ may *violate* one or more currently inactive bounds. (In this case, A_W cannot be the correct active set.) Feasibility is retained by determining the maximum nonnegative step $\alpha < 1$ such that $x + \alpha p$ is feasible. The bound that becomes satisfied exactly at $x + \alpha p$ is then "added" to the working set for the next iteration by adding a row of the identity matrix to A_W.

Otherwise, $x + p$ is feasible, and $\bar{x} = x + p$. Since p is the step to the minimizer of (1.3), it must hold that $Z^T g(\bar{x}) = 0$, which implies that $g(\bar{x}) = A_W^T \mu$ for some Lagrange multiplier vector μ. If the components of μ corresponding to active lower bounds are nonnegative, and those corresponding to active upper bounds are nonpositive, then $\bar{x}$ is the (unique) solution of (1.1). Otherwise, there is at least one component with the "wrong" sign, which means that deleting the corresponding constraint from the working set will produce a feasible direction of descent for the objective function. (The same interpretation applies if $p = 0$: either x is optimal for (1.1), or a constraint can be deleted from the working set.) When a bound constraint is deleted, the associated variable is said to be "freed" from its bound, and one of the rows of I_x is removed from A_W.

The standard convergence properties of this algorithm are summarized by the following two theorems, which are stated without proof (see, e.g., Gill and Murray, 1978; Fletcher, 1981).

Theorem 1.1. *(Linear independence of the working set). If the initial working set is chosen so that A_0 has full row rank, and if $A_W p = 0$ at all subsequent iterations, then: (i) every working set has full row rank; and (ii) the projected Hessian is positive definite at every iteration.* ∎

Theorem 1.2. *Assume that the feasible region of (1.1) has no degenerate vertices, i.e., the set of constraints defining every vertex is linearly independent. Then the feasible-point active-set method described above will terminate at the unique minimizer of (1.1) in a finite number of iterations.* ∎

If degenerate vertices exist, additional procedures should be included in the algorithm to prevent *cycling*, i.e., making an infinite number of changes in the working set without moving from the current point. Recent techniques for treating degeneracy are described in, for example, Fletcher (1985), Busovača (1985), Dax (1985), Osborne (1985), Ryan and Osborne (1986), and Gill *et al.* (1987b).

1.3. Special properties of the standard form. So far, we have discussed the role of the matrix A_W without particular attention to the computational advantages that arise when the problem is in standard form. Standard form allows the nonzero ("free") components of the

search direction to be computed using a matrix whose column dimension is equal to the number of free variables (rather than the total number of variables). To formalize this idea, let n_{FR} denote the number of free variables (i.e., corresponding to bounds not in A_W), and let the subscript "FR" denote the corresponding components of a vector or matrix. For example, A_{FR} denotes the $m \times n_{FR}$ submatrix of columns of A corresponding to free variables. Similarly, the subscript "FX" means the components corresponding to *fixed* variables (i.e., those whose bounds are in the working set). (We shall henceforth switch freely between the terminologies of "working sets" and "free/fixed variables". In general, the main iteration will be described in terms of changes to the working set because the structure of the constraints does not affect the algorithm at that level.)

The vector p_{FR} satisfies the linear system

$$\begin{pmatrix} H_{FR} & A_{FR}^T \\ A_{FR} & \end{pmatrix} \begin{pmatrix} -p_{FR} \\ \pi \end{pmatrix} \equiv K \begin{pmatrix} -p_{FR} \\ \pi \end{pmatrix} = \begin{pmatrix} g_{FR} \\ \end{pmatrix}, \tag{1.6}$$

where π denotes the vector of multipliers for the general equality constraints. The *reduced gradient* vector λ for the fixed variables (i.e., the Lagrange multiplier associated with active bounds) may be computed as $\lambda = \bar{g}_{FX} - A_{FX}^T \pi$, where $\bar{g}$ denotes $g(x + p)$.

Equation (1.6) is called the *Kuhn-Tucker* system or just the KT system. The following lemma characterizes the relationship between the eigenvalues of K and the eigenvalues of the projected Hessian $Z^T H Z$.

Lemma 1.1. *Let M be an $n \times n$ symmetric matrix and N an $m \times n$ matrix of full row rank. If Z is an $n \times (n - m)$ matrix such that $NZ = 0$, then*

$$\text{In} \begin{pmatrix} M & N^T \\ N & \end{pmatrix} = \text{In}(Z^T M Z) + (m, m, 0).$$

Proof. For a proof, see Gould (1985). ∎

The inertia of K can be deduced by applying Lemma 1.1 to (1.6) and invoking Theorem 1.1 to show that the projected Hessian is positive definite; thus, we conclude that $\text{In}(K) = (n, m, 0)$.

1.4. Special features of large quadratic programs. Techniques for obtaining a null-space basis Z explicitly or implicitly have been extensively studied recently, with particular reference to continuity (see, e.g., Coleman and Sorensen, 1984; Gill, Murray, Saunders, Stewart and Wright, 1985). When A is dense, Z is usually computed directly from a QR factorization of A. When A is sparse, however, known techniques for obtaining an orthogonal *and sparse* Z may be expensive in time and storage, although some recent approaches appear promising (see, e.g., Coleman and Pothen, 1986; Gilbert and Heath, 1986).

The representation of Z most commonly used in sparse problems is called the *reduced-gradient* form of Z, and is obtained as follows. The columns of A_{FR} are partitioned so as to identify explicitly an $m \times m$ nonsingular matrix B (the *basis*). Assuming that B is at the "left" of A_{FR}, we have

$$A_{FR} = (B \quad S). \tag{1.7}$$

(In practice, the columns of B may occur anywhere.)

When A_{FR} has the form (1.7), a basis for the null space of A_{FR} is given by the columns of the (non-orthogonal) matrix Z_{FR} defined as

$$Z_{FR} = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix}, \quad \text{so that} \quad Z = \begin{pmatrix} Z_{FR} \\ 0 \end{pmatrix}. \tag{1.8}$$

Furthermore,

$$Z_{FR}^T H_{FR} Z_{FR} = Z^T H Z.$$

Let $n_z = n - (m + n_{FX})$, so that Z_{FR} has n_z columns. The form of (1.8) means that matrix-vector products $Z^T v$ or Zv can be computed using a factorization of B (typically, a sparse LU factorization; see Gill *et al.*, 1987a), and Z need not be stored explicitly.

For large problems, the projected Hessian $Z^T H Z$ associated with the solution of (1.6) will generally be much denser than H and B. If n_z is small enough to allow the storage of a dense matrix of dimension $n_z \times n_z$, the null-space basis provided by (1.8) is very effective in methods that *approximate* $Z^T H Z$ (see Murtagh and Saunders, 1978).

2. The Schur-Complement Quadratic Programming Method

When A and H are large and sparse, a *single* system of the form (1.6) can be solved reliably and efficiently with the sparse matrix package MA27 (see Duff and Reid, 1982; Duff, Erisman and Reid, 1986). In an active-set QP method, however, a *sequence* of such systems must be solved, each differing from the preceding by a single row and column. In a method based on a straightforward interpretation of (1.6), the search direction p and multiplier μ would be computed from a KT system that varies in composition and size as the working set changes. In contrast, we now show that the special nature of these changes allows us to define a QP algorithm in which the solution of (1.6) may be obtained during k successive iterations using a *fixed* factorization of the initial KT system, and a factorization of a smaller matrix of (at most) order k.

2.1. Computation of the search vector and multipliers. To illustrate an iteration of the Schur-complement method, we first consider an example with 4 variables and a single general constraint, where bounds 2 and 4 are in the initial working set. Thus, $p_2 = p_4 = 0$, and the initial KT system (1.6) is

$$\begin{pmatrix} h_{11} & h_{13} & a_{11} \\ h_{13} & h_{33} & a_{13} \\ a_{11} & a_{13} & 0 \end{pmatrix} \begin{pmatrix} -p_1 \\ -p_3 \\ \pi_1 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_3 \\ 0 \end{pmatrix}. \tag{2.1}$$

At the next iteration, suppose that the first variable is to be fixed on a bound, so that $p_1 = 0$. It is easy to verify that p satisfying (1.6) for the revised working set satisfies

$$\left(\begin{array}{ccc|c} h_{11} & h_{13} & a_{11} & 1 \\ h_{13} & h_{33} & a_{13} & 0 \\ a_{11} & a_{13} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right) \begin{pmatrix} -p_1 \\ -p_3 \\ \pi_1 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_3 \\ 0 \\ 0 \end{pmatrix}, \tag{2.2}$$

where λ_1 is the reduced gradient for the newly fixed variable.

If, on the other hand, variable 2 is to be freed from its bound at the next iteration, the desired p satisfies

$$\left(\begin{array}{ccc|c} h_{11} & h_{13} & a_{11} & h_{12} \\ h_{13} & h_{33} & a_{13} & h_{23} \\ a_{11} & a_{13} & 0 & a_{12} \\ \hline h_{12} & h_{23} & a_{12} & h_{22} \end{array}\right) \begin{pmatrix} -p_1 \\ -p_3 \\ \pi_1 \\ -p_2 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_3 \\ 0 \\ g_2 \end{pmatrix}. \tag{2.3}$$

The general rule is that the previous KT system is bordered by a row and column of the identity when a variable is fixed, and by the free elements of a row and column of H and a column of A when a variable is freed.

The above process can be repeated in an obvious way over a *sequence* of iterations. Let x_0 denote the initial point of the sequence, K_0 the KT system at x_0, and n_0 the number of free variables at x_0. Assume that k changes to the working set have taken place since K_0 was factorized. Let f denote an $(n_0 + m)$-vector whose first n_0 components are the components of the current gradient corresponding to the free variables at x_0, and whose remaining m elements are zero. Let the k-vector w be defined as

$$w_j = \begin{cases} g_s(x) & \text{if } x_s \text{ was freed at iteration } j; \\ 0 & \text{otherwise.} \end{cases}$$

Note that both f and w depend on the current iterate x.

After k iterations, the symmetric bordered system to be solved is of dimension at most n_0+k, and has the form

$$\begin{pmatrix} K_0 & U \\ U^T & V \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ w \end{pmatrix}, \tag{2.4}$$

where U is $n_0 \times k$ and V is $k \times k$. The j-th column of U is a column of the identity if a variable was fixed at the j-th iteration; otherwise it contains elements from H and A, as described above. The nonzero entries of V are elements of H. (If no variables have been freed, U contains only columns of the identity and V is zero.)

The vectors y and z must be "unscrambled" to obtain p, μ and λ. The first n_0 components of y are the elements of $-p$ corresponding to the free variables at x_0. The remaining nonzero elements of p and the reduced gradients for the newly fixed variables are found from z as follows:

$$z_j = \begin{cases} \lambda_s & \text{if } x_s \text{ was fixed at iteration } j; \\ -p_s & \text{if } x_s \text{ was freed at iteration } j. \end{cases}$$

Since (2.4) (in general) increases in dimension by one at every iteration, it might appear that there is no benefit from this approach. However, the key point is that (2.4) can be solved using factorizations of K_0 and C, the $k \times k$ *Schur complement of* K_0:

$$C \equiv V - U^T K_0^{-1} U \tag{2.5}$$

(see, e.g., Bisschop and Meerhaus, 1977, 1980; Gill *et al.*, 1984). The following equations are solved in turn:

$$K_0 v = f \tag{2.6a}$$

$$Cz = w - U^T v \tag{2.6b}$$

$$K_0 y = f - Uz. \tag{2.6c}$$

Thus, the work required to perform a QP iteration is dominated by two solves with K_0 and one solve with C. If k is small enough, dense QR or LU factors of C may be maintained. To exploit symmetry, the symmetric indefinite factorization (see Section 5) must be used. Each of these factorizations can be updated efficiently and in a numerically stable manner to reflect changes in the working set; see, e.g., Gill *et al.* (1974), Gill *et al.* (1987a) and Sorensen (1977). In all cases, the numerical stability of (2.6) depends largely on the condition of K_0.

Each change in the working set results in addition of a new row and column to C. To show this, consider a *single* change in the working set, and write the associated KT system as

$$\begin{pmatrix} K_0 & \bar{U} \\ \bar{U}^T & \bar{V} \end{pmatrix}, \quad \text{where} \quad \bar{U} = \begin{pmatrix} U & u \end{pmatrix} \quad \text{and} \quad \bar{V} = \begin{pmatrix} V & v \\ v^T & \sigma \end{pmatrix}. \tag{2.7}$$

(The definitions of u, v and σ depend on the nature of the change in the working set.) The new Schur complement $\bar{C}$ for (2.7) is given by

$$\bar{C} = \bar{V} - \bar{U}^T K_0^{-1} \bar{U} = \begin{pmatrix} V & v \\ v^T & \sigma \end{pmatrix} - \begin{pmatrix} U^T \\ u^T \end{pmatrix} K_0^{-1} \begin{pmatrix} U & u \end{pmatrix}. \tag{2.8}$$

Comparison of (2.7) and (2.5) reveals that the Schur complement is bordered by a single row and column:

$$\bar{C} = \begin{pmatrix} C & t \\ t^T & \gamma \end{pmatrix}, \tag{2.9a}$$

where

$$K_0 q = u, \qquad t = v - U^T q \quad \text{and} \quad \gamma = \sigma - u^T q. \tag{2.9b}$$

Note that a solve with K_0 is needed to update C.

The dimension of C need not increase if a variable returns to its original status during the sequence of iterations. For example, suppose that a given variable is fixed at the initial point, subsequently freed, and then later fixed again at either the same or opposite bound. (The same comments apply if a variable is originally free, and then fixed and freed again.) To effect the second change in the working set, the dimension of C can be *reduced* by one, by simply removing the column of U associated with the first change and then modifying C to "undo" the first update. (It is easy to show that deleting a column from U is equivalent to deleting a row and column from C.)

In Section 5.2 we identify the special relationship of the Schur complement to the projected Hessian in certain cases. It is therefore of interest to know the inertia of the Schur complement, which is characterized by the following lemma.

Lemma 2.1. *Consider an iteration of a feasible-point active-set method in which p and μ are computed from (2.6). If i_{FX} of the fixed variables were originally free at x_0, and i_{FR} of the free variables were originally fixed at x_0, then*

$$\text{In}(C) = (i_{FR}, i_{FX}, 0).$$

Proof. The bordered matrix (2.4) may be permuted (symmetrically) to a matrix M of the form

$$M = \begin{pmatrix} G & A^T & E^T \\ A & & \\ E & & \end{pmatrix},$$

where E is formed from i_{FX} rows of the identity matrix. The elements of G and A are formed from variables in two categories: those that were free at x_0 (and therefore in K_0) and those that were freed in subsequent iterations. Let ν denote the dimension of G.

The inertia of $\bar{C}$ (2.9) after a single change in the working set is given by

$$\text{In}(\bar{C}) = \text{In}(C) + \text{In}(\bar{C}/C),$$

and we may also write

$$\text{In}(\bar{C}/C) = \text{In}(\bar{M}/M) = \text{In}(\bar{M}) - \text{In}(M).$$

Since the projected Hessian matrices $Z^T H Z$ and $Z^T H \bar{Z}$ are positive definite, Lemma 1.1 implies that

$$\text{In}(\bar{M}) - \text{In}(M) = (\bar{\nu}, m + \bar{i}_{FX}, 0) - (\nu, m + i_{FX}, 0).$$

If M is expanded by fixing a variable on its bound, the dimension of G remains the same, but E is expanded by a single row (a coordinate vector). Therefore, $\bar{\nu} = \nu$, $\bar{i}_{FX} = i_{FX} + 1$ and

$$\text{In}(\bar{C}) = \text{In}(C) + (0, 1, 0).$$

Similarly, if a variable is freed from its bound, $\bar{\nu} = \nu + 1$, $\bar{i}_{FX} = i_{FX}$ and

$$\text{In}(\bar{C}) = \text{In}(C) + (1, 0, 0).$$

The desired result follows by applying the arguments above to each expansion of the Schur complement. ∎

2.2. Refactorization. As the dimension of C grows, the work needed to solve (2.6) increases, as does the required storage. It is therefore necessary to "restart" at a "new" x_0, and to factorize the current KT system from scratch (as in linear programming, where the current basis is refactorized.)

Typically, the dimension of C is allowed to reach a specified limit (say, 100) before refactorization. If the QP is a "later" subproblem in an SQP method, the solution is likely to be found before refactorization is required. The exact point at which refactorization becomes worthwhile depends on the problem. In general, the decision to refactorize is guided by considerations similar to those in the simplex method for linear programming—i.e., it is probably desirable to refactorize when the cost of an iteration exceeds an average figure determined by amortizing the cost of the initial factorization over a number of iterations. Refactorization may also be mandated by a lack of storage.

Refactorization provides an opportunity to check for any possible deterioration in feasibility through the accumulation of rounding errors, by computing the row residuals $b - Ax_0$ for the general constraints. If x_0 is unacceptable because of large *row errors* (i.e., large residuals in the general constraints), one or two steps of iterative refinement may be helpful (see, e.g., Wilkinson, 1963; Björck, 1987). Unfortunately, iterative refinement can cause some of the variables to violate their bounds. It is therefore essential for any application of iterative improvement to include a procedure for restoring feasibility with respect to the bounds (see Section 4). Because of rounding errors, the possibility of cycling during this process cannot be completely eliminated. For example, the algorithm could cycle forever between points that alternately violate the bounds and general

constraints. However, the changes to the variables caused by refinement tend to be small, and cycling is unlikely.

Ill-conditioning in K_0 may lead to serious error in the computed solution of the bordered system (2.4). However, as each new variable is fixed on its bound, the KT system may become better conditioned and merit refactorization. In some circumstances, refactorization can be postponed by applying iterative refinement on both K_0 and C whenever Ap has drifted away from zero. (This form of iterative refinement is unlikely to cause loss of feasibility with respect to the bounds.) Another alternative is to use deflated block elimination (see Chan, 1984; Chan and Grossi, 1985).

3. Avoiding a Solve with K_0

When solving (2.4), the calculations can be rearranged so that only *one* solve with K_0 is needed at every iteration (in contrast to the two solves in (2.6)). The "trick" is to define the right-hand side of (2.4) so that certain components *do not change*. This can be accomplished by computing the step q from x_0 to the minimizer of (1.3), rather than the step p from x, i.e., q satisfies

$$x_0 + q = x + p.$$

To illustrate this process, we reconsider the four-variable example of Section 2.1. Suppose that the next iteration involves fixing variable 1 at its lower bound ℓ_1. To make the first component of $x_0 + q$ equal to ℓ_1, q must satisfy

$$\left(\begin{array}{ccc|c} h_{11} & h_{13} & a_{11} & 1 \\ h_{13} & h_{33} & a_{13} & 0 \\ a_{11} & a_{13} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right) \begin{pmatrix} -q_1 \\ -q_3 \\ \pi_1 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} g_1(x_0) \\ g_3(x_0) \\ 0 \\ (x_0 - \ell)_1 \end{pmatrix},$$

where $g(x_0)$ denotes the quadratic objective gradient at x_0. It is easy to verify that the first component of $x_0 + q$ is given by

$$(x_0 + q)_1 = (x_0)_1 + q_1 = (x_0)_1 - (x_0 - \ell)_1 = \ell_1,$$

as required.

On the other hand, if variable 2 is to be freed at the next iteration, then q satisfies

$$\left(\begin{array}{ccc|c} h_{11} & h_{13} & a_{11} & h_{12} \\ h_{13} & h_{33} & a_{13} & h_{23} \\ a_{11} & a_{13} & 0 & a_{12} \\ \hline h_{12} & h_{23} & a_{12} & h_{22} \end{array}\right) \begin{pmatrix} -q_1 \\ -q_3 \\ \pi_1 \\ -q_2 \end{pmatrix} = \begin{pmatrix} g_1(x_0) \\ g_3(x_0) \\ 0 \\ g_2(x_0) \end{pmatrix}.$$

To carry out this strategy, let f_0 denote the vector f from Section 2.1 *evaluated at* x_0, and let the k-vector w be defined as

$$w_j = \begin{cases} (x_0 - x)_s & \text{if } x_s \text{ was fixed at iteration } j; \\ g_s(x_0) & \text{if } x_s \text{ was freed at iteration } j. \end{cases}$$

The value $(x_0 - x)_s$ is defined as w_j when variable s is fixed at iteration k because x_s will then equal ℓ_s or u_s (depending on which bound is in the working set).

In general, given f_0 and w, the search direction and multiplier vector may be found from the solution of

$$\begin{pmatrix} K_0 & U \\ U^T & V \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f_0 \\ w \end{pmatrix}. \tag{3.2}$$

Examination of (2.6*a*) shows that the right-hand side associated with (3.2) is *constant*, so that the solution of $K_0 v_0 = f_0$ needs to be computed only once, at the first iteration. Thereafter, (2.6*b*) is simply

$$Cz = w - U^T v_0.$$

Only one of the properties mentioned in Section 2 does not apply to this modified iteration. The exception occurs when a variable moves from one bound to its "opposite" bound. In this case, it is not possible to decrease the dimension of the Schur complement and maintain a constant vector f_0, since deleting the column of U has the effect of fixing the variable at the *original* bound instead of the "new" bound. To allow for this special case, the Schur complement is expanded as if the variable had originally been free at x_0. This modification is a special case of (2.9) with the values $u = 0$ and $v = e_s$, where s denotes the iteration at which the variable became free, thereby adding the s-th row and column to C. The new Schur complement is given by

$$\bar{C} = \begin{pmatrix} C & e_s \\ e_s^T & \end{pmatrix},$$

and it is not necessary to compute q, t or γ in (2.9).

4. Finding an Initial Feasible Point

In order to apply the active-set algorithm described in Section 1.3, a feasible starting point is necessary. In the dense case, problem (1.1) is often solved in two phases. The first (the *feasibility phase*) finds a feasible point by minimizing the sum of infeasibilities; the second (the *QP phase*) minimizes the quadratic objective function in the feasible region.

In a null-space method for large quadratic problems, the following procedure can be used to find a feasible point. Given a basis B (a nonsingular $m \times m$ submatrix of A), a point x_0 can computed such that $Ax_0 = b$, and then tested for feasibility with respect to the bounds. If some of the bounds are violated, a direction can be computed that strictly decreases the sum of bound violations, yet remains "on" the general constraints. Once a variable satisfies its bounds, it is not allowed to become infeasible in subsequent iterations. In a typical null-space method, both the feasibility and QP phases use the same factorizations, and the two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function (see Gill, Murray, Saunders and Wright, 1985).

Unfortunately, this approach will be inefficient within a QP algorithm based on direct solution of the KT system, since the factors of B cannot be used to initiate the QP phase. The inefficiency is even more serious when the QP is a "later" subproblem within an SQP method, since a feasible point for the QP can usually be obtained directly from knowledge of the active set in the previous subproblem. If no iterations are required to find a feasible point, the effort required to factorize

the basis would be wasted. Similar inefficiences occur if ill-conditioning in the QP phase causes a loss of feasibility.

To avoid these difficulties, the feasibility phase can be modified so that it attempts to reduce the objective function while simultaneously improving feasibility. The objective function in the feasibility phase then becomes a *composite* objective function (a weighted sum of the infeasibilities and the original quadratic objective function). With this approach, the search direction and multiplier vector satisfy an KT system similar to (1.6) in *both* phases. The major difference between the feasibility and QP iterations is that the steplength α in (1.2) is restricted to ensure that the *number* of violated bounds does not increase.

An alternative strategy for the feasibility phase has been suggested in the single-phase methods of Hoyle (1986). In this case, x_0 is chosen to satisfy the bound constraints, and each search direction satisfies a system of the form

$$\begin{pmatrix} H_{FR} & A_{FR}^T \\ A_{FR} & \end{pmatrix} \begin{pmatrix} -p_{FR} \\ \pi \end{pmatrix} = \begin{pmatrix} g_{FR} \\ r \end{pmatrix}, \tag{4.1}$$

where $r = Ax - b$. Unless r in (4.1) is zero, the general constraints are not satisfied. As soon as a step $\alpha = 1$ is taken, r becomes zero, and the iterates thereafter satisfy all the constraints.

Neither of these approaches is completely satisfactory for the SC method. With a composite objective function, the gradient vector changes in a discrete fashion as variables become feasible and so must be recomputed at each iteration, which makes it impossible to save the solve with K_0 during the feasibility phase (see Section 3). If we apply the approach based on solving (4.1), Theorem 1.1 does not apply as long as r is nonzero, and fixing a variable may cause A_{FR} to become rank-deficient. In this situation, special procedures must be invoked to maintain full rank of the working set.

A composite objective function that seems well suited to the SC method involves solving a QP subproblem with an additional variable ξ (the *artificial variable*) associated with the infeasibilities. As in the approach of Hoyle, the bound constraints are always satisfied, so that the procedure may begin with any x_0 satisfying $\ell \le x_0 \le u$. Let r_0 denote the residual $b - Ax_0$, let the unit vector s be defined by $s = r_0/\|r_0\|$, and let $\xi_0 = 1$. Given a positive weight ρ, we solve the modified quadratic program

$$\begin{aligned} &\underset{x,\xi}{\text{minimize}} \quad \rho\xi + c^T x + \tfrac{1}{2} x^T H x \\ &\text{subject to} \quad \begin{pmatrix} A & s \end{pmatrix} \begin{pmatrix} x \\ \xi \end{pmatrix} = b, \quad \ell \le x \le u, \quad 0 \le \xi \le 1, \end{aligned} \tag{4.2}$$

for which (x_0, ξ_0) is feasible. If the artificial variable ever becomes zero, a feasible point has been found, and ξ is thereafter excluded from the problem.

With formulation (4.2), full rank of the initial working set implies full rank in all subsequent working sets, so that no special procedures are needed to correct for rank deficiency. Further, the gradient of the composite objective function of (4.2) is a smooth function of x. If ρ is sufficiently large, the solution of (4.2) is identical to that of the original problem (1.1).

Note that the "artificial column" s is almost always dense. Consequently, ξ_0 is treated as initially fixed on its upper bound so that it is not included in K_0. If ρ is large enough, it is

possible to ensure that ξ will be the first variable to be freed. The reduced cost for the upper bound on ξ is $\rho - s^T\pi$, which exceeds the reduced cost for any remaining non-optimal variable if

$$\rho > s^T\pi + \max\{\,|\hat{g}_j - a_j^T\pi| : j \in \mathcal{J}\}, \tag{4.3}$$

where $\hat{g} = g(x_0 + p)$ and $\mathcal{J}$ is the index set of non-optimal fixed variables. The lower bound in (4.3) may be used as an initial estimate of ρ.

If the artificial variable ever moves to its lower bound immediately after being freed, the Schur complement (a single element) is discarded. This situation often occurs when the artificial variable is introduced to rectify infeasibility during the QP phase.

In any method that relies on a composite objective function, a strategy must be included to attempt to decide whether no feasible point exists for the original problem (1.1), or whether ρ has not yet become sufficiently large. (No such strategy can ever be guaranteed.) The value of ρ is typically increased if $\xi > 0$ at the solution of (4.2)—for example, ρ could be multiplied by a factor—but eventually the algorithm must "give up" and declare that the constraints of (1.1) appear to be infeasible. A strategy that increases ρ gradually may be inefficient if only a single QP is solved, since many values of ρ may be required. When a sequence of related problems is solved, however, the value of ρ from one QP is usually a satisfactory choice for the next.

If no feasible point exists, it is often desirable to locate the minimium sum of infeasibilities. Although the solution of (4.2) does minimize a weighted sum of infeasibilities if ρ is large enough, the weights are essentially arbitrary (since they depend on the initial point).

5. Computing the Initial Factorization

The matrix K_0 is represented by its *symmetric indefinite factorization* (see, e.g., Bunch and Parlett, 1971, and Bunch and Kaufman, 1977):

$$K_0 = LDL^T, \tag{5.1}$$

where L is lower triangular and D is block diagonal, with 1×1 or 2×2 blocks. (The latter are required to retain numerical stability.)

An effective and widely used implementation of (5.1) for sparse matrices is the Harwell routine MA27 (Duff and Reid, 1982), which is a three-phase method. The ANALYZE phase is purely symbolic (i.e., uses only the sparsity pattern of K_0), and applies a version of the minimum-degree algorithm intended to define a symmetric ordering that produces low fill-in in L. In the subsequent FACTORIZE phase, the numerical factors (5.1) are computed using the actual entries in K_0 ordered as prescribed by ANALYZE, with further symmetric interchanges performed if necessary for numerical stability. Finally, the solution of $K_0 x = b$ is computed in the SOLVE phase.

5.1. A specialized ANALYZE phase. The direct application of MA27 is very effective for problems in which H is sparse and there are few general constraints (i.e., m is small relative to n). Many problems in statistics have this feature, since they require nonnegativity of all variables, with the single general constraint $\sum_i x_i = 1$. KT systems with small m are best handled by applying the ANALYZE phase to the matrix H only. A suitable ordering for the full KT system may then be determined by expanding the data structure to include the constraint

rows and columns. If all the 1 × 1 pivots are numerically acceptable in the FACTORIZE phase, the first search direction and multiplier vector would satisfy the *range-space equations*:

$$A_{FR}^T H_{FR}^{-1} A_{FR} \pi = A_{FR}^T H_{FR}^{-1} g_{FR} \quad \text{and} \quad H_{FR} p_{FR} = A_{FR}^T \pi - g_{FR} \tag{5.2}$$

(see, e.g., Gill *et al.*, 1982). In practice, some 1 × 1 pivots may be rejected—for example, any pivot corresponding to a free slack variable. However, if m is not too large, the symbolic ordering for H_{FR} is still likely to provide a numerically stable factorization.

When m is not small relative to n, the rows of A_{FR} must be included in the ANALYZE phase. Unfortunately, the minimum-degree algorithm assumes that no 2 × 2 pivots occur during the factorization, and hence that the diagonal elements of K_0 are nonzero. Since K_0 *always* has a zero diagonal block in the lower right-hand corner, the symbolic ordering from ANALYZE is often changed substantially during the FACTORIZE phase. In some cases, the resulting additional fill-in increases the work required to operate with the factors. The problem is exacerbated if zero diagonal elements occur *within* H—for example, the diagonals corresponding to slack variables.

In our experience with large m, an increase in fill-in compared to the predicted level has occurred consistently, even when all the diagonals of H are nonzero. A possible explanation is that the number of nonzeros in a given row of A_{FR} is likely to be less than the total number of nonzeros in a given row of H_{FR} and the corresponding column of A_{FR}. Consequently, the minimum-degree algorithm may well choose an ordering with many zero diagonals. The root of the difficulty seems to be the persistence of zero diagonal pivots in the reduced matrix.

The special treatment of zero diagonal elements during the minimum-degree ordering will be incorporated in a new version of MA27 (Duff, private communication). However, the efficiency of the current version of MA27 on KT systems may be significantly improved in some cases by utilizing the facility of MA27 to accept a preassigned ordering for the FACTORIZE. Let T denote a 2 × 2 matrix (called a *tile*) of the form

$$T_{ij} = \begin{pmatrix} h & a \\ \bar{a} & \end{pmatrix}, \tag{5.3}$$

where h is an element of H, and a and $\bar{a}$ are elements of A. A symmetric tile has the useful property that it is nonsingular if a is nonzero; moreover, its nonzero eigenvalues must have opposite sign. Our ordering strategy is to define a permutation matrix Π such that the upper left-hand corner of a symmetrically permuted version of K_0 consists of a symmetric "checkerboard" matrix T of tiles, i.e.,

$$M = \Pi^T K_0 \Pi = \begin{pmatrix} T & F^T \\ F & E \end{pmatrix}, \tag{5.4}$$

where T has the form

$$T = \begin{pmatrix} T_{11} & T_{12} & T_{13} & \cdots \\ T_{12}^T & T_{22} & T_{23} & \cdots \\ T_{13}^T & T_{23}^T & T_{33} & \cdots \\ \vdots & \vdots & & \end{pmatrix}.$$

Each tile (5.3) is essentially a pairing of a column of H with a column of A. For example, when T is 4×4, one possible arrangement is

$$T = \left(\begin{array}{cc|cc} h_{11} & a_{21} & h_{13} & a_{11} \\ a_{21} & & a_{23} & \\ \hline h_{13} & a_{23} & h_{33} & a_{13} \\ a_{11} & & a_{13} & \end{array}\right).$$

Once an element a_{ij} is selected for a diagonal tile T_{qq}, the elements of column j of A become entries of the off-diagonal tiles T_{pq}, where $p = 1, 2, \ldots, q-1$. The remaining elements in row i of A become ineligible for inclusion in diagonal tiles.

In the proposed method, the sparsity pattern of M in (5.4) is first processed by a minimum-degree ordering that treats each tile in T as a *single element* that is nonzero if and only if the tile is a nonzero matrix. A suitable ordering for the full matrix is then determined by expanding the data structure to consider each tile as a 2×2 matrix. The idea is to force MA27 to use a pivot strategy contrary to that typically used in the Bunch-Parlett algorithm—i.e., instead of choosing 2×2 pivots only when no suitable diagonal pivots are available, 2×2 pivots are *preferred*.

Arranging the Hessian and general constraints of (1.6) into tiles is a very effective method for dealing with slack variables. If all free slacks are picked first for inclusion in the diagonal tiles, the associated 2×2 pivots cause no fill-in during the symmetric indefinite factorization.

Many real-world problems have the desirable feature that a "natural" pairing of columns of H and A is suggested by the nature of the underlying physical system. By applying the above technique in the ANALYZE phase, fill-in during the factorization may be substantially reduced compared to an application of MA27 alone. In Table 1 we give some factorization statistics obtained from applying the tiling strategy to solve QP subproblems arising in the optimal distribution of electrical power (Burchett, Happ and Vierath, 1984). For each problem we give the dimension of the projected Hessian, the dimension of the KT system, the number of nonzeros in K_0, the number of elements in L predicted by the ANALYZE, and the actual number of nonzeros generated during the FACTORIZE.

Table 1
Factorization statistics for various KT systems

$\dim(K_0)$	$\dim(Z^T H Z)$	K_0	ANALYZE	FACTORIZE
1840	56	13593	33486	34676
2048	266	17025	37614	38458
2105	219	13858	38696	38700
5841	1	31697	63278	69323
5997	157	33399	65170	69620

There is a clear need for a *general* procedure that will find a suitable tiling for an arbitrary sparse KT system. Given any $2m \times 2m$ tiled matrix T, there exists a permutation $\bar{\Pi}$ such that

$$\bar{T} = \bar{\Pi}^T T \bar{\Pi} = \begin{pmatrix} H & B^T \\ B & \end{pmatrix}, \tag{5.5}$$

where B is an $m \times m$ subset of the columns of A. Since A has full rank, there must exist permutations Π and $\bar{\Pi}$ for which B is nonsingular. Hence there must exist at least one tiling such that T is nonsingular. Finding a "good" set of columns from A is very similar to the problem of choosing a nonsingular basis. Because the selected columns of A automatically define the rows and columns of H to be used in T, the "best" arrangement will minimize the number of nonzero tiles. Once an initial T matrix has been chosen, updating T should be relatively easy when refactorization is required. In particular, no change is necessary if none of the free variables corresponding to the selected columns of H has become fixed. This property implies that construction of the tiles should favor columns of H that are likely to remain free.

5.2. Relationship between methods. If the KT system is solved by taking pivots from H_{FR} first, the symmetric indefinite factorization implicitly forms matrices that define the class of range-space methods (*cf.* (5.2)). The following theorem shows that a different pivot order will cause the symmetric indefinite factorization to form matrices associated with the reduced-gradient method.

Lemma 5.1. *Define two matrices M and $\bar{M}$ such that*

$$M = \begin{pmatrix} G & F^T \\ F & E \end{pmatrix} \quad \text{and} \quad \bar{M} = \begin{pmatrix} \bar{G} & \bar{F}^T \\ \bar{F} & E \end{pmatrix},$$

where G and $\bar{G}$ are $k \times k$ and nonsingular, and $\bar{M}$ is obtained from M by performing symmetric permutations of the first k rows and columns. Then $\bar{M}/\bar{G} = M/G$. ∎

Theorem 5.1. *Assume without loss of generality that H_{FR} and A_{FR} may be partitioned so that*

$$H_{FR} = \begin{pmatrix} H_1 & G^T \\ G & H_2 \end{pmatrix} \quad \text{and} \quad A_{FR} = \begin{pmatrix} B & S \end{pmatrix},$$

where B and H_1 are $m \times m$ with B nonsingular. Let T denote a $2m \times 2m$ tiled matrix formed from elements of H_1 and B. If M denotes the permuted KT system (5.4) then

$$M/T = Z^T H Z, \quad \text{where} \quad Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}.$$

Proof. Let M and T denote the matrices

$$M = \begin{pmatrix} H_1 & B^T & G \\ B & & S \\ G^T & S^T & H_2 \end{pmatrix} \quad \text{and} \quad T = \begin{pmatrix} H_1 & B^T \\ B & \end{pmatrix}.$$

Then by definition,

$$M/T = H_2 - \begin{pmatrix} G^T & S^T \end{pmatrix} T^{-1} \begin{pmatrix} G \\ S \end{pmatrix} = H_2 - \begin{pmatrix} G^T & S^T \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}, \tag{5.6}$$

where V_1 and V_2 are defined by the (block-triangular) equations

$$\begin{pmatrix} H_1 & B^T \\ B & \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} G \\ S \end{pmatrix}. \tag{5.7}$$

Substituting for V_1 and V_2 from (5.7) into (5.6), we obtain

$$\begin{aligned} M/T &= H_2 - G^T B^{-1} S - S^T B^{-T} G + S^T B^{-T} H_1 B^{-1} S \\ &= \begin{pmatrix} -S^T B^{-T} & I \end{pmatrix} \begin{pmatrix} H_1 & G^T \\ G & H_2 \end{pmatrix} \begin{pmatrix} -B^{-1} S \\ I \end{pmatrix} = Z^T H Z. \end{aligned}$$

The result now follows from Lemma 5.1. ∎

This result implies that the symmetric indefinite factorization, when used in conjunction with a nonsingular tiling, *implicitly* forms and factorizes the projected Hessian. Moreover, $Z^T H Z$ is computed in a way that exploits symmetry. In situations when the projected Hessian is sparse, the symmetric indefinite factorization thus provides an effective means of exploiting sparsity during the factorization.

In general, direct factorization of the KT system provides the opportunity to define a set of methods for the large-scale case, with each method determined by the order of the columns in the initial KT factorization.

6. Formulation of the Constraints

In this section we discuss two aspects of the occasional inefficiency resulting from use of the standard form (1.1).

6.1. Treatment of slacks in the standard form. In the Schur-complement algorithm, each *free slack column adds a unit row and column to* K_0, so that symmetric interchanges cannot move the corresponding unit element to the diagonal. In order to avoid unnecessary fill-in during the initial factorization, the two (unit) nonzeros associated with each slack must be formed into a 2×2 tile as described in Section 5. If there are k free slacks, A_{FR} is of the form

$$A_{FR} = \begin{pmatrix} & A_1 \\ I_k & A_2 \end{pmatrix}, \tag{6.1}$$

where I_k is the identity of order k. With suitable permutation, the associated KT system has the form

$$K_0 = \begin{pmatrix} & I_k & & \\ I_k & & A_2 & \\ & A_2^T & H_1 & A_1^T \\ & & A_1 & \end{pmatrix}. \tag{6.2}$$

A trivial sequence of interchanges in the leading $2k \times 2k$ rows and columns of (6.2) gives the required tiling, with k matrices

$$\begin{pmatrix} & 1 \\ 1 & \end{pmatrix}$$

on the principal diagonal. If 2×2 pivots are selected from the diagonal, no additional fill-in occurs in the leading $2k$ columns. Ideally, a symmetric indefinite solver such as MA27 should treat singleton rows and columns in this way.

6.2. General inequalities. Constraints sometimes arise "naturally" in the form

$$\ell \le \begin{pmatrix} I \\ A \end{pmatrix} x \le u. \tag{6.3}$$

In an active-set method based on (6.3), the working-set matrix undergoes both row *and* column changes. For sparse problems, few authors have considered the associated complications of updating sparse factors that vary in dimension. (See Gill *et al.*, 1987a, for an exception.)

In contrast, the Schur-complement method may be generalized quite readily to problems with constraints of the form (6.3). The equations associated with each iteration are identical to (1.6), except that A_{FR} is effectively just the submatrix A_1 in (6.1). If a general constraint with gradient a is added to the working set, the KT system is bordered by a vector made up from the free elements of a.

The update for deletion of a general constraint from the working set can be illustrated easily for the first iteration. If the constraint to be deleted corresponds to the s-th row of A_{FR}, the search direction satisfies the bordered system

$$\begin{pmatrix} H_{FR} & A_{FR}^T & \\ A_{FR} & & e_s \\ & e_s^T & \end{pmatrix} \begin{pmatrix} -p_{FR} \\ \pi \\ -\gamma \end{pmatrix} = \begin{pmatrix} g_{FR} \\ \\ \end{pmatrix}.$$

where γ may be discarded. The Lagrange multipliers for the general constraints in the working set may be recovered by deleting the s-th element of π.

Since A_{FR} must be maintained subject to both row and column updates, it is necessary to access A by both rows and columns. In the case of linear programming, if $m \ll n$ the standard form is efficient (and requires only column updates), while if $m \gg n$ one may prefer to solve the *dual problem*, again in standard form. However, once the form (6.3) is assumed, efficiency can be retained regardless of the ratio of m to n. This advantage is all the more important for nonlinear problems, where the device of solving the dual is not necessarily applicable or efficient.

7. Discussion

An important feature of the Schur-complement approach is that any advances in methods for sparse linear equations are immediately applicable to computation of the initial factorization of K_0. This approach is especially effective when K_0 has special structure that can be exploited in a "black box" equation solver--e.g., when the constraints are derived from network flow problems.

Many new machines have become available in recent years with vector and/or parallel capabilities. In most cases, the novelty of their architecture is not exploited by existing software. In the large-scale mathematical programming area, a portable Fortran code (e.g., MINOS, SCICONIC) will run successfully on vector machines like the CRAY-1 or CRAY-XMP, but most of the computation will be performed only in scalar mode.

It can be expected that sparse linear equation solvers will eventually be developed for novel machines intended for scientific computation. While explicit updating of LU factors will probably remain efficient on conventional machines (see Gill *et al.*, 1987a), the Schur-complement approach is likely to provide the most effective method for machines with advanced architectures. In the case of vector supercomputers, techniques have already been developed that allow large linear systems to be solved efficiently (see Ashcraft *et al.*, 1987). We therefore believe that the efficient solution of large quadratic programming problems on vector machines is now feasible.

The Schur-complement method described here has been implemented within an SQP method and applied to the solution of optimal power flow (OPF) problems arising in the electrical power

industry. OPF problems concern the optimal generation and distribution of electrical power in a network (see Stott, Alsac and Marinho, 1980). Exact second derivatives are available for these problems, and a specialized Newton-based SQP method has been applied with great success to general OPF problems of a size previously considered intractable.

Acknowledgements

We would like to thank Rob Burchett for numerous discussions on large-scale SQP methods. He also provided the motivation for considering the Schur-complement method. We are grateful to Nicholas Higham for his careful reading of the manuscript.

References

Ashcraft, C. C., Grimes, R. G., Lewis, J. G., Peyton, B. W. and Simon, H. D. (1987). Recent progress in sparse matrix methods for large linear systems on vector supercomputers, Engineering Technology Applications Report ETA-TR-56, Boeing Computer Services, Seattle, Washington.

Best, M. J. (1984). Equivalence of some quadratic programming algorithms, *Mathematical Programming* 30, 71–87.

Bisschop, J. and Meeraus, A. (1977). Matrix augmentation and partitioning in the updating of the basis inverse, *Mathematical Programming* 13, 241–254.

Bisschop, J. and Meeraus, A. (1980). Matrix augmentation and structure preservation in linearly constrained control problems, *Mathematical Programming* 18, 7–15.

Björck, Å. (1987). Iterative refinement and reliable computing, in M. G. Cox and S. J. Hammarling (eds.), *Advances in Reliable Numerical Computing*, Oxford University Press.

Bunch, J. R. and Kaufman, L. C. (1977). Some stable methods for calculating inertia and solving symmetric linear equations, *Mathematics of Computation* 31, 163–179.

Bunch, J. R. and Parlett, B. N. (1971). Direct methods for solving symmetric indefinite systems of linear equations, *SIAM J. on Numerical Analysis* 8, 639–655.

Bunch, J. R. and Kaufman, L. C. (1980). A computational method for the indefinite quadratic programming problem, *Linear Algebra and its Applications* 34, 341–370.

Burchett, R. C., Happ, H. H. and Wirgau, K. (1982). Large-scale optimal power flow, *IEEE Transactions on Power Apparatus and Systems*, 3722–3732.

Busovača, S. (1985). Handling degeneracy in a nonlinear ℓ_1 algorithm, Technical Report CS-85-34, University of Waterloo, Waterloo, Canada.

Chan, T. F. (1984). Deflated decomposition of solutions of nearly singular systems, *SIAM J. on Numerical Analysis* 21, 738–754.

Chan, T. F. and Grossi, T. A. (1985). DBEPACK: A program package for solving bordered singular systems, Technical Report YALEU/DCS/RR-336 Yale University.

Coleman, T. F. and Pothen, A. (1986). The null space problem I. Complexity, *SIAM J. on Algebraic and Discrete Methods* 7, 527–537.

Coleman, T. F. and Sorensen, D. C. (1984). A note on the computation of an orthogonal basis for the null space of a matrix, *Mathematical Programming* 29, 234–242.

Cottle, R. W. (1974). Manifestations of the Schur complement, *Linear Algebra and its Applications* 8, 189–211.

Dax, A. (1985). The computation of descent directions at degenerate points, Technical report, Hydrological Service, PO Box 6381, Jerusalem, Israel.

Djang, A. (1980). *Algorithmic Equivalence in Quadratic Programming*, Ph.D. Thesis, Stanford University, California.

Duff, I. S., Erisman, A. M. and Reid, J. K. (1986). *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, New York.

Duff, I. S. and Reid, J. K. (1982). MA27—a set of Fortran subroutines for solving sparse symmetric sets of linear equations, Report AERE R-10533, Computer Science and Systems Division, AERE Harwell, Oxfordshire, England.

Duff, I. S. and Reid, J. K. (1984). The multifrontal solution of unsymmetric sets of linear equations, *SIAM J. on Scientific and Statistical Computing* 5, 633–641.

Fletcher, R. (1981). *Practical Methods of Optimization, Volume 2, Constrained Optimization*, John Wiley and Sons, New York and Toronto.

Fletcher, R. (1985). Degeneracy in the presence of round-off errors, Numerical Analysis Report NA89, Department of Mathematical Sciences, University of Dundee, Scotland.

Gilbert, J. R. and Heath, M. T. (1986). Computing a sparse basis for the null space, Report TR86-730, Department of Computer Science, Cornell University, Ithaca, New York.

Gill, P. E., Golub, G. H., Murray, W. and Saunders, M. A. (1974). Methods for modifying matrix factorizations, *Mathematics of Computation* 28, 505–535.

Gill, P. E., Gould, N. I. M., Murray, W., Saunders, M. A. and Wright, M. H. (1982). Range-space methods for convex quadratic programming, Report SOL 82-14, Department of Operations Research, Stanford University.

Gill, P. E. and Murray, W. (1978). Numerically stable methods for quadratic programming, *Mathematical Programming* 14, 349–372.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1984). Sparse matrix methods in optimization, *SIAM J. on Scientific and Statistical Computing* 5, 562–589.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1985). Software and its relationship to methods, in P. T. Boggs, R. H. Byrd and R. B. Schnabel (eds.), *Numerical Optimization 1984*, SIAM, Philadelphia, 139–159.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1987a). Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications* 88/89, 239–270.

Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H. (1987b). A practical anti-cycling procedure for linear and nonlinear programming, Report SOL 87-13, Department of Operations Research, Stanford University.

Gill, P. E., Murray, W., Saunders, M. ..., Stewart, G. W. and Wright, M. H. (1985). Properties of a representation of a basis for the null space, *Mathematical Programming* 33, 172–186.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London and New York.

Gould, N. I. M. (1985). On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem, *Mathematical Programming* 32, 90–99.

Hoyle, S. C. (1986). A single-phase method for quadratic programming, Report SOL 86-9, Department of Operations Research, Stanford University.

Murtagh, B. A. and Saunders, M. A. (1978). Large-scale linearly constrained optimization, *Mathematical Programming* 14, 41–72.

Osborne, M. R. (1985). *Finite Algorithms in Optimization and Analysis*, John Wiley, Chichester and New York.

Ryan, D. M. and Osborne, M. R. (1986). On the solution of highly degenerate linear programs, Technical report, Department of Statistics, Australian National University.

Sorensen, D. C. (1977). Updating the symmetric indefinite factorization with applications in a modified Newton method, Report ANL-77-49, Argonne National Laboratory, Argonne, Illinois.

Stott, B., Alsac, O. and Marinho, J. L. (1980). The optimal power flow problem, in A. M. Erisman, K. W. Neves and M. H. Dwarakanath (eds.), *Electric Power Problems: The Mathematical Challenge*, SIAM, Philadelphia, 327–351.

Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem*, Oxford University Press.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 87-12	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Schur-Complement Method for Sparse Quadratic Programming		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip E. Gill, Walter Murray, Michael A. Saunders, Margaret H. Wright		8. CONTRACT OR GRANT NUMBER(s) N00014-87-K-0142, AFOSR-87-01962
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1111MA
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE October 1987 13. NUMBER OF PAGES 20 pp.
Air Force Office of Scientific Research/NM Building 410 Bolling Air Force Base Washington, DC 20332		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

large-scale quadratic programming, supercomputers, constrained optimization, sparse matrix methods, Schur complement.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(please see other side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

In applying active-set methods to sparse quadratic programs, it is desirable to utilize existing sparse-matrix techniques. We describe a quadratic programming method based on the classical Schur complement. Its key feature is that much of the linear algebraic work associated with an entire sequence of iterations involves a fixed sparse factorization. Updates are performed at every iteration to the factorization of a smaller matrix, which may be treated as dense or sparse.

The use of a fixed sparse factorization allows an "off-the shelf" sparse equation solver to be used repeatedly. This feature is ideally suited to problems with structure that can be exploited by a specialized factorization. Moreover, improvements in efficiency derived from exploiting new parallel and vector computer architectures are immediately applicable.

An obvious appliation of the method is in sequential quadratic programming methods for nonlinearly constrained optimization, which require solution of a sequence of closely related quadratic programming subproblems. We discuss some ways in which the known relationship between successive problems can be exploited.

END

DATE

FILMED

3-88

DTIC